



ТЕХНИЧЕСКИ
УНИВЕРСИТЕТ - СОФИЯ



КУРСОВА РАБОТА

ПО

СИНТЕЗ И АНАЛИЗ НА АЛГОРИТМИ

НА ТЕМА:

„Система за известяване за съмнителна активност“

Изготвил:
Лъчезар Христов Спириев

Проверил/ла:

Специалност:
Телекомуникации

София, 2022

Условие на задачата
Система за известяване за съмнителна активност

Националната банка има полица да предупреждава клиенти за възможна съмнителна активност в техните сметки. Ако сумата изхарчена от клиента в даден ден е по-голяма или равна на 2x от средната сума в продължение на няколко дни, изпращат известие за потенциална измама. Банката не изпраща известия на клиента, ако не засекат повишена активност поне няколко дни подред.

Даден е броят на поредни дни **d** и дневните разходи на клиента за период от n дни, намерете и запишете броят пъти, в който клиентът ще получи известие по време на тези **n** дни.

На пример, **d=3** и разходите = [10,20,30,40,50]. За първите три дни банката събира информация за разходите. На 4-тия ден сме наблюдавали последователни разходи от [10,20,30]. Средния разход е 20, а на 4-тия ден са похарчени 40. Понеже **40>2x20**, клиентът ще получи известие. На следващия ден последователните разходи са [20,30,40] и изхарчените са 50. Това е по-малко от **2x30** , затова няма да бъде изпратено известие. По време на наблюдавания период бе изпратено само едно известие.

Note: Средната стойност на поредица от числа се намира като подредим всички числа от най-малко към най-голямо. Ако броят на числата в поредицата е нечетен, средното число се избира като средната стойност. Ако броят на числата е четен, взимаме средната стойност на двете числа по средата на поредицата. (Wikipedia)[1]

[1] Median

Решение на задачата

```
def insertionSort(arr):

    for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
            arr[j + 1] = key
    return arr


def sortedInsert(arr, n):

    start = 0
    end = len(arr) - 1

    while (start < end):
        mid = (start + end)//2
        if n >= arr[mid]:
            start = mid+1
        else:
            end = mid-1
    if n <= arr[start]:
        arr.insert(start, n)
    else:
        arr.insert(start+1, n)
    return arr

def sortedRemove(arr, n):

    start = 0
    end = len(arr) - 1

    while (start <= end):
        mid = (start + end)//2
        if n == arr[mid]:
            arr.pop(mid)
            return arr
        if n > arr[mid]:
            start = mid+1
        else:
            end = mid-1
```

```python
def activityNotifications(expenditure, d):

    count = 0
    m = d//2

    for i in range(len(expenditure)-d):

        if i == 0:
            temp = expenditure[:d]
            insertionSort(temp)
        else:
            expense_insert = expenditure[i+d-1]
            temp = sortedInsert(temp, expense_insert)

        current_expense = expenditure[i+d]
        twice_median = temp[m] + temp[-(m+1)]
        next_remove = expenditure[i]

        if current_expense >= twice_median:
            count += 1
        temp = sortedRemove(temp, next_remove)

    return count

if __name__ == '__main__':

    n = int(input("Enter the number of days: "))

    d = int(input("Enter the number of trailing days: "))
    expenditure = []

    for i in range(0, n):
        a = int(input("Enter the expenditure: "))
        expenditure.append(a)


    print("Number of received notifications: ",
activityNotifications(expenditure, d))
```

Изпълнение на задачата

Първият ред съдържа цяло число n, броят на дните, за които имаме данни за транзакциите, а вторият ред - броят на дните, които следим за редния разход на клиента. Редовете под тях са следните разходи на клиента и последният ред показва броят на получените известия.

```
Enter the number of days: 7
Enter the number of trailing days: 4
Enter the expenditure: 20
Enter the expenditure: 10
Enter the expenditure: 50
Enter the expenditure: 90
Enter the expenditure: 20
Enter the expenditure: 10
Enter the expenditure: 40
Number of received notifications:  0
```

```
Enter the number of days: 10
Enter the number of trailing days: 6
Enter the expenditure: 20
Enter the expenditure: 10
Enter the expenditure: 50
Enter the expenditure: 20
Enter the expenditure: 30
Enter the expenditure: 80
Enter the expenditure: 100
Enter the expenditure: 30
Enter the expenditure: 80
Enter the expenditure: 40
Number of received notifications:  2
```

Обяснение на задачата

Обикновено обясненията следва да започнат от първият ред на това което четем, но в програмирането не е точно така. За да разберем точно как работи нашата задача, нека започнем от нейният ред на работа. Проследяването по този начин ни довежда от това да започнем от линията `if __name__ == '__main__':`, този ред означава че нашата задача ще работи ако я пуснем като „script", но няма да работи ако бъде импортната като модул, но това е нещо в което няма да навлизаме, защото за нас това няма особено значение, но добре разделя и показва от къде всъщност започва кодът. Променливата „n" която виждаме се използва за въвеждането на всички дни в които са се извършвали транзакции от потребителя, променливата „d" служи за проследяване на разходите на дните на потребителя за избран интервал от време. „Expenditure" е масивът, в който ние ще въведем всички стойности похарчени от потребителя в избраните от него дни. Въвеждането започва от първия ден и това се осъществява от цикълът `for i in range(0, n):`, в него ние задаваме стойности на променливата „a" която след това тя бива добавена към масива с „append". След като вече имаме необходима информация, ние излизаме от цикълът и отиваме на командата „print", която се обръща към функцията „activityNotifications" и не откарва до следваща част от нашия код.

Функцията „activityNotifications" може да бъде наречена главна функция, защото чрез нея се достъпват и използват всички останали функции – „sortedRemove", „sortedInsert" и „insertionSort". Нека да видим как се случва това. Тази функция започва с инициализация отново на две променливи, но този път те не са въведени от потребителя. Променливата „count" служи за отброяването на известията които потребителят ще получи на краят на задачата, докато променливата „m" служи за едно изчисление което ще бъде направено малко по-нататък в нашия код. След като инициализираме двете променливи влизаме в цикълът `for i in range(len(expenditure)-d):`, който служи за обхождане на масивът въведен от потребителят, но този път само за броят на дните избрани за проследяване. След като влезнем в този цикъл имаме проверка `if i == 0:`, която винаги ще бъде вярна първоначално и ни позволява да влезнем в следващата наша функция „insertionSort". В тази проверка първо приравняваме нашият масив на нов масив наречен „temp" който ще обработим, за да не променяме подредбата на оригиналният.

Вече сме подали новият ни масив на функцията с която ще го обработваме. Първото нещо което се случва в тази функция е да влезем отново в цикъл който започва от втория елемент и продължава до краят на масива. След това инициализираме две променливи „key" и „j", променливата „key" я приравняваме винаги да има същата стойност като елементът на масива върху който се намираме в момента, докато променливата „j" я приравняваме винаги да сочи на 1 място преди „key" което означава на предходния елемент в масива. След което ние влизаме в още един цикъл `while j >= 0 and key < arr[j]:`, който ще продължи докато променливата „j" е по-голяма от 0 и докато променливата „key" е по-малка от предходният елемент в масива. Когато тези две изисквания са изпълнение елементът който се намира на позицията пред нашия елемент „j" придобива неговата стойност а след което старото място в масивът на променливата „j" бива запълнено от стойността записана в променливата „key". Ние знаем обаче че тези два елемента се намират един до друг, което означава, че те реално си разменят местата докато променливата „key" има по-голяма стойност от

променливата „j“, но написано малко по-сложно в код. След като тези действия се повторят необходимият брой пъти и масивът бъде подреден ние връщаме новият подреден масив на нашата функция „insertionSort“ и се връщаме обратно в главната функция.

След като направихме нашата проверка, отиваме на следващите ни три инициализирани променливи – „current_expense“, „twice_median“, „next_remove“. Променливата „expenditure“ приема за стойността си елементът който се намира на една позиция след избраните от потребителя дни за проследяване, тоест стойността която ни трябва за да сравним дали разходите за този ден са два пъти по-големи или не от средно аритметичната стойност от дните избрани от потребителя за проследяване. За да намерим стойността която е два пъти по-голяма от средно аритметичната за избран период от дни използваме следващата променлива „twice_median“. Която ако имаме четен брой елементи зима средната стойност в масива и я събира със себе си, или ако имаме нечетна стойност на елементите в масива зима двата средни елемента които са един до друг и ги събира. И стигаме до последната променлива „next_remove“ която взима първия елемент от нашия оригинален масив и придобива неговата стойност. Като създадем всички тези променливи имаме проверка дали разходите в „current_expense“ са по-големи или равни на средно аритметичните разходи в „twice_median“, ако това е вярно нотификациите се покачват с 1, ако не те остават непроменени, след което отиваме в следващата ни функция „sortedRemove“.

Функцията „sortedRemove“ отново започва с инициализация на две променливи „start“ и „end“. “Start” е равна на 0, докато „End“ е винаги с 1 по-малко от размера на масива „temp“, след това влизаме в цикъл `while (start <= end):`, в който правим изчисление в което винаги получаваме позицията на средният елемент в масива и го приравняваме на променливата „mid“. След което правим проверка дали нашето число което зададохме още в „next_remove“ съвпада с числото в масива което се намира в центъра. Ако това е така числото бива изтрито и получаваме новият масив, но ако това не е вярно, проверяваме дали числото е по-голямо от средното число в масива, ако това е така проверяваме дали следващото по-голямо число е това което търсим или не е, ако отново не е това което търсим продължаваме да сравняваме с всяко нарастващо число докато не го открием. При първата проверка обаче ако нашето число е по-малко от средното число в масива, правим отново същото но за всяко по-малко число докато не открием това което търсим, след което изтриваме първо число което съвпадне с това което търсим.

Сега започва последната част от нашия код. Вече изтрих ме първият елемент от масивът след като приключихме с него. Сега е време да вземем новият масив и да се върнем в главната ни функцията „activityNotifications“ , където този път не влизаме в проверката `if i == 0, а в проверката под нея else:`, където „expense_insert“ придобива стойността на „current_expense“ и тя бива добавена в масивът за проследяване чрез функцията „sortedInsert“. Като инициялизация и начин на работа тази функция работи по почти един и същ начин с функцията „sortedRemove“ единствената разлика, е че не търсим конкретна стойност за изтриване, а просто сравняваме със средната стойност в масива и проверяваме дали тя е по-голяма или по-малка от нашата взета стойност. Ако нашата стойност е по-голяма от средната продължаваме сравняването с всяка следваща стойност по-голяма от средната в масива, докато намерим стойност по-голяма от нашата взета и я поставим преди нея.

Обяснение на използваният алгоритъм в задачата – Insertion Sort

Сортирането чрез вмъкване работи подобно на сортирането на карти за игра в ръце. Предполага се, че първата карта вече е сортирана в играта с карти и след това избираме несортирана карта. Ако избраната несортирана карта е по-голяма от първата карта, тя ще бъде поставена от дясната страна; в противен случай ще бъде поставен от лявата страна. По същия начин всички несортирани карти се вземат и поставят на точното им място.

Същият подход се прилага при сортиране чрез вмъкване. Идеята зад сортирането чрез вмъкване е, че първо вземете един елемент, интегрирайки го през сортирания масив. Въпреки че е лесен за използване, той не е подходящ за големи набори от данни, тъй като времевата сложност на сортирането при вмъкване в средния и най-лошия случай е O(n2), където n е броят на елементите. Сортирането чрез вмъкване е по-малко ефективно от другите алгоритми за сортиране.

Сортирането чрез вмъкване има различни предимства като –

Просто изпълнение
Ефективен за малки набори от данни
Адаптивен, т.е. подходящ е за набори от данни, които вече са значително сортирани.

Сложност на сортиране на вмъкване

Сега, нека видим времевата сложност на сортирането на вмъкване в най-добрия случай, средния случай и в най-лошия случай. Ще видим и пространствената сложност на сортирането чрез вмъкване.

1. Времева сложност

Времева сложност на случая:

Най-добър случай - O(n)
Среден случай - $O(n^2)$
Най-лошия случай - $O(n^2)$

Сложност в най-добрия случай - възниква, когато не е необходимо сортиране, т.е. масивът вече е сортиран. Най-добрият случай на времева сложност на сортиране при вмъкване е O(n).

Средна сложност на случая - възниква, когато елементите на масива са в разбъркан ред, който не е правилно възходящ и неправилно низходящ. Средната времева сложност на случай на сортиране при вмъкване е O(n2).

Сложност в най-лошия случай - възниква, когато се изисква елементите на масива да бъдат сортирани в обратен ред. Това означава, че трябва да сортирате елементите на масива във възходящ ред, но неговите елементи са в низходящ ред. Най-лошият случай на времева сложност на сортиране при а вмъкване е O(n2).[2]

[2] javaTpoint

Използвани източници

Bibliography

javaTpoint. (n.d.). *insertion sort*. Retrieved from javatpoint.com: https://www.javatpoint.com/insertion-sort

Median. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Median#Basic_procedure